AFRL-OSR-VA-TR-2015-0109

TOOLS FOR RAPID UNDERSTANDING OF MALWARE CODE

Saumya Debray
ARIZONA UNIV BOARD OF REGENTS TUCSON

05/07/2015
Final Report

Air Force Research Laboratory
AF Office Of Scientific Research (AFOSR)/ RTC
Arlington, Virginia 22203
Air Force Materiel Command

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
04-05-2015	Final	15-07-2011 - 14-03-2015

4. TITLE AND SUBTITLE
Tools for Rapid Understanding of Malware Code

5a. CONTRACT NUMBER
n/a

5b. GRANT NUMBER
FA9550-11-1-0191

5c. PROGRAM ELEMENT NUMBER
n/a

6. AUTHOR(S)
Debray, Saumya K

5d. PROJECT NUMBER
n/a

5e. TASK NUMBER
n/a

5f. WORK UNIT NUMBER
n/a

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Department of Computer Science
The University of Arizona
Tucson, AZ 85721

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
University of Arizona
888 N Euclid Ave
Tucson, AZ 85719-4824

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for Public Release; Distribution is Unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
A significant shortcoming of existing malware analysis tools is their lack of general-purpose automated support for dealing with advanced code obfuscation techniques. Computer malware are developing increasingly sophisticated techniques to thwart analysis, and the lack of such automated tool support significantly increases the extent of manual intervention necessary for extracting and understanding what the malware is doing. Such intervention is tedious and time-consuming, and has the effect of reducing the speed with which new malware threats can be addressed. This is a serious problem because swift and precise response is essential in order to combat cyber-attacks in a timely and effective manner. This project aims to address the lack of automated tool support for malware analysis by developing a general framework and techniques to automate much of the task of deobfuscating malware binaries and thereby dramatically speed up the process of understanding malware code. This is done through two main objectives: the development of semantics-based techniques for identifying and removing obfuscation code; and the synthesis of simplification techniques to transform the resulting low-level machine code to program representations that are easier to reason about and understand.

15. SUBJECT TERMS
Cyber-security; malware analysis; software obfuscation.

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Debray, Saumya K
Unclassified	Unclassified	Unclassified	UU		19b. TELEPHONE NUMBER *(Include area code)* 520-621-4527

Reset

INSTRUCTIONS FOR COMPLETING SF 298

1. REPORT DATE. Full publication date, including day, month, if available. Must cite at least the year and be Year 2000 compliant, e.g. 30-06-1998; xx-06-1998; xx-xx-1998.

2. REPORT TYPE. State the type of report, such as final, technical, interim, memorandum, master's thesis, progress, quarterly, research, special, group study, etc.

3. DATES COVERED. Indicate the time during which the work was performed and the report was written, e.g., Jun 1997 - Jun 1998; 1-10 Jun 1996; May - Nov 1998; Nov 1998.

4. TITLE. Enter title and subtitle with volume number and part number, if applicable. On classified documents, enter the title classification in parentheses.

5a. CONTRACT NUMBER. Enter all contract numbers as they appear in the report, e.g. F33615-86-C-5169.

5b. GRANT NUMBER. Enter all grant numbers as they appear in the report, e.g. AFOSR-82-1234.

5c. PROGRAM ELEMENT NUMBER. Enter all program element numbers as they appear in the report, e.g. 61101A.

5d. PROJECT NUMBER. Enter all project numbers as they appear in the report, e.g. 1F665702D1257; ILIR.

5e. TASK NUMBER. Enter all task numbers as they appear in the report, e.g. 05; RF0330201; T4112.

5f. WORK UNIT NUMBER. Enter all work unit numbers as they appear in the report, e.g. 001; AFAPL30480105.

6. AUTHOR(S). Enter name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. The form of entry is the last name, first name, middle initial, and additional qualifiers separated by commas, e.g. Smith, Richard, J, Jr.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES). Self-explanatory.

8. PERFORMING ORGANIZATION REPORT NUMBER. Enter all unique alphanumeric report numbers assigned by the performing organization, e.g. BRL-1234; AFWL-TR-85-4017-Vol-21-PT-2.

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES). Enter the name and address of the organization(s) financially responsible for and monitoring the work.

10. SPONSOR/MONITOR'S ACRONYM(S). Enter, if available, e.g. BRL, ARDEC, NADC.

11. SPONSOR/MONITOR'S REPORT NUMBER(S). Enter report number as assigned by the sponsoring/ monitoring agency, if available, e.g. BRL-TR-829; -215.

12. DISTRIBUTION/AVAILABILITY STATEMENT. Use agency-mandated availability statements to indicate the public availability or distribution limitations of the report. If additional limitations/ restrictions or special markings are indicated, follow agency authorization procedures, e.g. RD/FRD, PROPIN, ITAR, etc. Include copyright information.

13. SUPPLEMENTARY NOTES. Enter information not included elsewhere such as: prepared in cooperation with; translation of; report supersedes; old edition number, etc.

14. ABSTRACT. A brief (approximately 200 words) factual summary of the most significant information.

15. SUBJECT TERMS. Key words or phrases identifying major concepts in the report.

16. SECURITY CLASSIFICATION. Enter security classification in accordance with security classification regulations, e.g. U, C, S, etc. If this form contains classified information, stamp classification level on the top and bottom of this page.

17. LIMITATION OF ABSTRACT. This block must be completed to assign a distribution limitation to the abstract. Enter UU (Unclassified Unlimited) or SAR (Same as Report). An entry in this block is necessary if the abstract is to be limited.

Final Report: Award No. FA9550-11-1-0191
Tools for Rapid Understanding of Malware Code

May 4, 2015

Executive Summary

The project goals were to develop automated techniques and tools for analysis and reverse engineering of highly-obfuscated malware codes. The project made significant progress in this regard. Two very different kinds of malware were considered; because of the different nature of the malicious code in each case, fundamentally different techniques were employed. The two kinds of malware, and specific accomplishments for each, are listed below.

1. *Web-delivered malware*, which is typically in the form of (obfuscated) JavaScript programs. This kind of malware is currently among the commonest way for infections to occur.

   Specific accomphishments include:

   (a) Development of novel techniques for reverse engineering obfuscated JavaScript code [2].

   (b) Identification of weaknesses in existing techniques for detecting web-borne malware [3, 5].

   (c) Development of client-side defenses against trigger-based web-based malware [4].

2. *Native executables*, whcich refer to machine code programs that execute natively. Regardless of whether an infection happened through the web via JavaScript code or not, in the end the malicious actions are typically carried out by native executables.

   Specific accomplishments include:

   (a) Development of improved techniques for information flow analysis in software [8].

   (b) Generic techniques for deobfuscation of executable code [9].

   (c) Information flow based dynamic analysis techniques for identifying self-checksum-based anti-tamper defenses in software [6, 7].

The project led to two PhD dissertations:

1. Gen Lu. *Analysis of Evasion Techniques in Web-Based Malware.* PhD Dissertation, The University of Arizona, Jan. 2014.

   This dissertation focused on analysis of web-based malware. The significant accomplishments of this work are listed above.

2. Babak Yadegari. *A Generic Approach to Deobfuscation*. PhD Dissertation, The University of Arizona, May 2016 (expected).

   This dissertation focused on analysis of native malware. The significant accomplishments of this work are listed above.

Software and data samples resulting from the project are available to the research community at `http://www.cs.arizona.edu/projects/lynx/Samples/`.

1 Project Objectives

This project aimed to address the lack of automated tool support for malware analysis by developing a general framework and techniques to automate much of the task of deobfuscating malware binaries and thereby dramatically speed up the process of understanding malware code. This goal was to be attained through two main objectives: first, the development of semantics-based techniques for identifying and removing obfuscation code; and second, the synthesis of simplification techniques to transform the resulting low-level machine code to program representations that are easier to reason about and understand.

2 Accomplishments/New Findings

The project focused on advanced semantics-based techniques to understand the behavior of obfuscated code, in particular code that may have been obfuscated in various ways to resist analysis, possibly using obfuscations that we do not know about and therefore cannot anticipate. In the context of this focus, the project looked at three main topics: analysis of web-borne malware, in particular drive-by downloads from infected web pages; analysis of executables armored with various static and dynamic anti-analysis defenses; and foundational topics in semantics-based malware analysis.

1. Analysis of Web-Borne Malware. Web-based mechanisms, often mediated by malicious JavaScript code, play an important role in malware delivery today, making defenses against web-borne malware crucial for system security. This work investigates improved techniques for defending against web-borne malware.

1. *Novel techniques for reverse engineering obfuscated JavaScript code* [2].

   Javascript is a scripting language that is commonly used to create sophisticated interactive client-side web applications. It can also be used to carry out browser-based attacks on users. Malicious JavaScript code is usually highly obfuscated, making detection a challenge. This work describes a simple approach to deobfuscation of JavaScript code based on dynamic analysis and slicing. Experiments using a prototype implementation indicate that the approach described is able to penetrate multiple layers of complex obfuscations and extract the core logic of the computation.

   Figure 1 shows an fragment of obfuscated JavaScript code we extracted from a malicious web page along with the deobfuscated code obtained automatically from it using our techniques. The original malware sample goes through three execution contexts: Context 1 resides in the web page opened by user's web browser; it is a small piece of obfuscated JavaScript code that, when executed, invokes document.write() method to dynamically insert a hidden iFrame, and causes an external web page to be loaded; in this case this web page was on a hacked web page in Germany. This newly loaded web page contains more obfuscated code (context 2). Context 2 then causes another level of code unfolding using eval() and generates context 3, which is the intended payload: this context uses a dynamically created hidden iFrame to open a PDF file, hosted on a machine in China, that exploits a vulnerability in Adobe Reader. The final recovered code is very close to what one might obtain if deobfuscating the malicious code manually; importantly, the intermediate steps involving web page redirections through

```
laKKs='mCha';jJt='';n9gs="37G51G67G105G102G114G97G109G1

...6 lines deleted...

7G105G102G114G97G109G101G37G51G69";ngs=document;n9gs=
n9gs["split"]('G');for(i=0;i<n9gs.length;i++)
jJt+=String['fro'+laKKs+'rCode'](n9gs[i]);
ngs["w"+"rite"](unescape(jJt));
```

(*a*) Original obfuscated JavaScript

```
var0 = 0;
while (var0 < navigator.plugins.length) {
  var1 = navigator.plugins[local_var0].name;
  if (var1.indexOf("Adobe Reader") != -1)
    document.write("<iframe src='./f3256c.pdf'
                    width='1' height='1')
    frameborder=0></iframe>");
  var0++;
  continue;
}
```

(*b*) Deobfuscated JavaScript

Figure 1: Semantics-based deobfuscation of malicious JavaScript sample

dynamic iFrames are removed during the deobfuscation process, leaving only the essence of the malicious actions.

2. *Identification of weaknesses in existing techniques for detecting web-borne malware* [3, 5].

   This work explores weaknesses in existing approaches to the detection of malicious JavaScript code. These approaches generally fall into two categories: lightweight techniques focusing on syntactic features such as string obfuscation and dynamic code generation; and heavier-weight approaches that look for deeper semantic characteristics such as the presence of shellcode-like strings or execution of exploit code. We show that each of these approaches has its weaknesses, and that state-of-the-art detectors using these techniques can be defeated using cloaking techniques that combine emulation with dynamic anti-analysis checks.

   Figure 2 shows the high-level architecture of software using these cloaking techniques. We used three malware detectors, covering a wide spectrum of detection technologies, for our experiments: VirusTotal, an online portal to a collection of anti-virus software with up-to-date exploit databases that exemplifies current commercial malware detection technology; Zozzle, a static detector based on machine learning; and Wepawet, a hybrid detection system based on JSAND that represents a state-of-the-art combination of static and dynamic analyses. These three detectors, range from traditional signature matching to state-of-the-art static and dynamic analyses, represent the current state of detection techniques. None of these detectors was able to penetrate the cloaking technique described and identify potentially malicious content embedded within the programs.

3. *Client-side defenses against trigger-based web-based malware* [4].

   Web-based malware tend to be environment-dependent, which poses a significant challenge on defending web-based attacks, because the malicious code—which may be exposed and activated only under specific environmental conditions such as the version of the browser—may not be triggered during analysis. This work proposes a simple approach for defending environment-dependent malware. Instead of increasing analysis coverage in detector, the goal of this technique is to ensure that the client will take the same execution path as the one examined by the detector. This technique is designed to work alongside a detector, it can handle cases existing multi-path exploration techniques are incapable of, and provides an efficient way to identify discrepancies in a JavaScript program's execution behavior in a user's environment compared to its behavior in a sandboxed detector, thereby detecting false negatives that may have been caused by environment dependencies. Experiments show that this technique can effectively detect environment-dependent behavior discrepancy of various forms, including those seen in real malware.

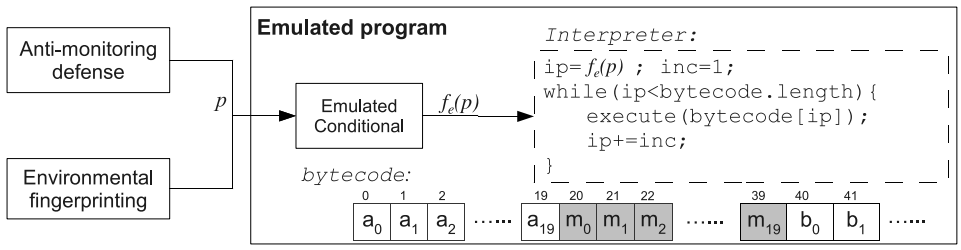
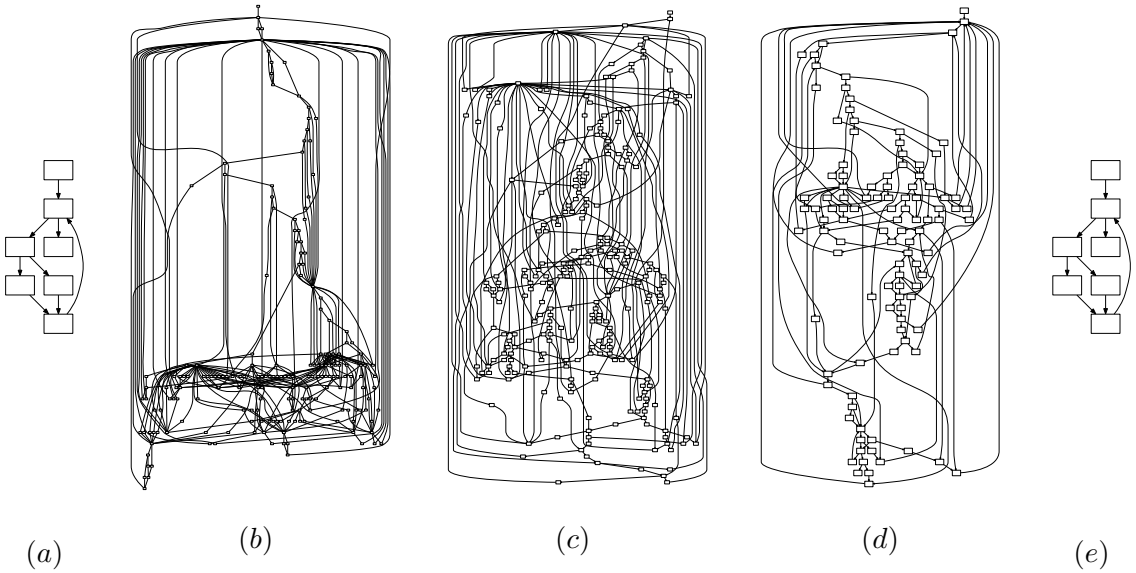


Figure 2: Code architecture to bypass (existing) defenses against web-borne malware



Key:

(a) Original program

(b) Obfuscated program

(c) Deobfuscation result: traditional byte-level taint analysis

(d) Deobfuscation result: bit-level analysis (taintedness information only)

(e) Deobfuscation result: enhanced bit-level analysis (taintedness + taint source information) (our algorithm)

Figure 3: Impact of different taint analysis algorithms on quality of debofuscation (Input program: binary search; obfuscated using: ExeCryptor)

2. Analysis of Obfuscation and Anti-Analysis Defenses in Executable Programs. Malicious software are usually armored in various ways to avoid detection and resist analysis. When new malware is encountered, such anti-analysis defenses have to be penetrated in order to understand the internal logic of the code and devise countermeasures. This work investigates various automatic and general-purpose ways for defeating anti-analysis and code obfuscation defenses.

1. *Improved techniques for information flow analysis in software* [8].

   Taint analysis has a wide variety of applications in software analysis, making the precision of taint analysis an important consideration. Current taint analysis algorithms, including previous work on bit-precise taint analyses, suffer from shortcomings that can lead to significant loss of precision (under/over tainting) in some situations. This work explores these limitations of existing taint analysis algorithms, shows how they can lead to imprecise taint propagation, and describes a generalization of current bit-level taint analysis techniques to address these problems and improve their precision. Experiments using a deobfuscation tool indicate that our enhanced taint analysis algorithm leads to significant improvements in the quality of deobfuscation.

   Figure 3 illustrates the improvement in the quality of reverse-engineering obfuscated malicious code using our algorithm compared to other taint analysis algorithms described in the literature.

2. *Generic techniques for deobfuscation of executable code* [9].

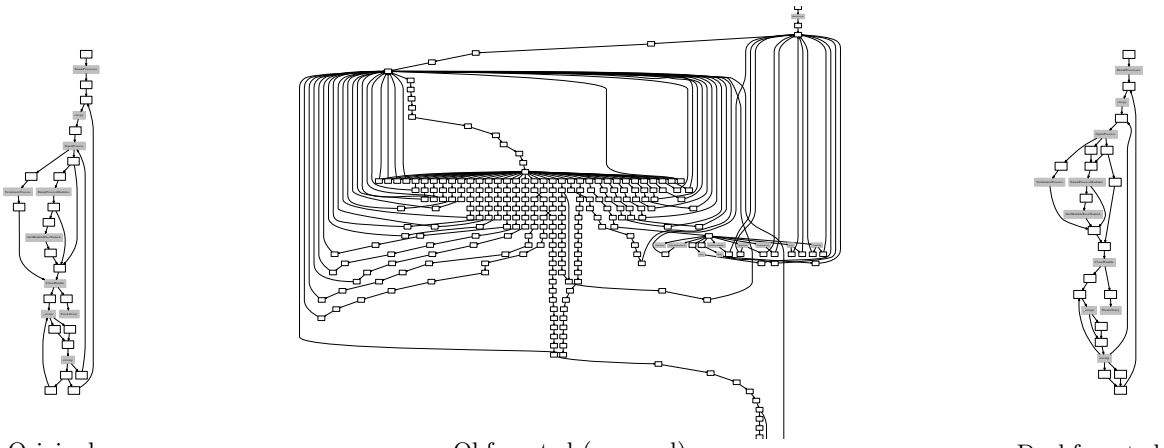
   This work discusses a generic approach for deobfuscation of obfuscated executable code. The approach described does not make any assumptions about the nature of the obfuscations used, but instead uses semantics-preserving program transformations to simplify away obfuscation code. We have applied a prototype implementation of our ideas to a variety of different kinds of obfuscation, including emulation-based obfuscation, emulation-based obfuscation with runtime code unpacking, and return-oriented programming. Our experimental results are encouraging and suggest that this approach can be effective in extracting the internal logic from code obfuscated using a variety of obfuscation techniques, including tools such as Themida that previous approaches could not handle.

   Figure 4 shows the effects of deobfuscation on several emulation-obfuscated malware samples.

3. *Information flow based dynamic analysis techniques for identifying self-checksum-based anti-tamper defenses in software* [6, 7].

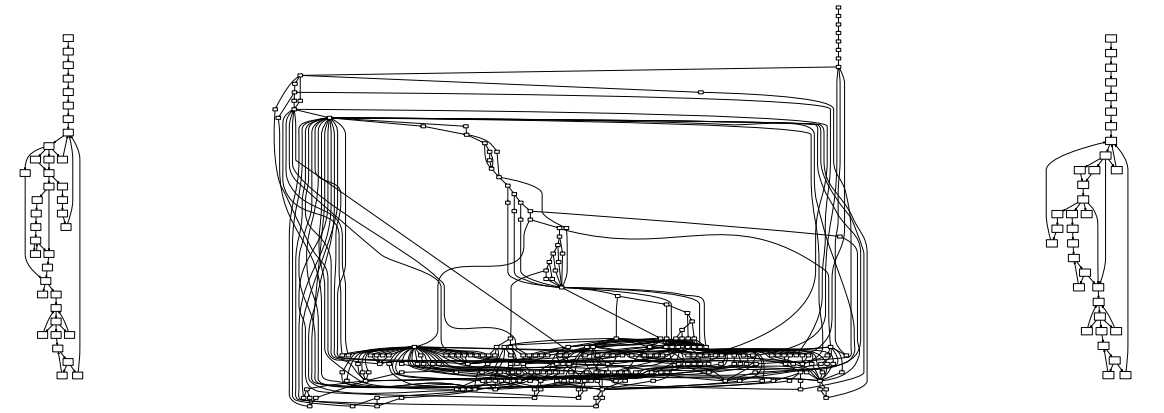
   Software self/checksumming is widely used as an anti/tampering mechanism for protecting intellectual property and deterring piracy. This makes it important to understand the strengths and weaknesses of various approaches to self-checksumming. This work investigates a dynamic information-flow-based attack that aims to identify and understand self-checksumming behavior in software. Our approach is applicable to a wide class of self-chesumming defenses and the information obtained can be used to determine how the checksumming defenses may be bypassed. Experiments using a prototype implementation of our ideas indicate that our approach can successfully identify self-checksumming behavior in (our implementations of) proposals from the research literature.

3. Semantics-based Approaches to Malware Analysis. This work uses the theoretical framework of *abstract interpretation* to investigate foundational issues in semantics-based malware detection.



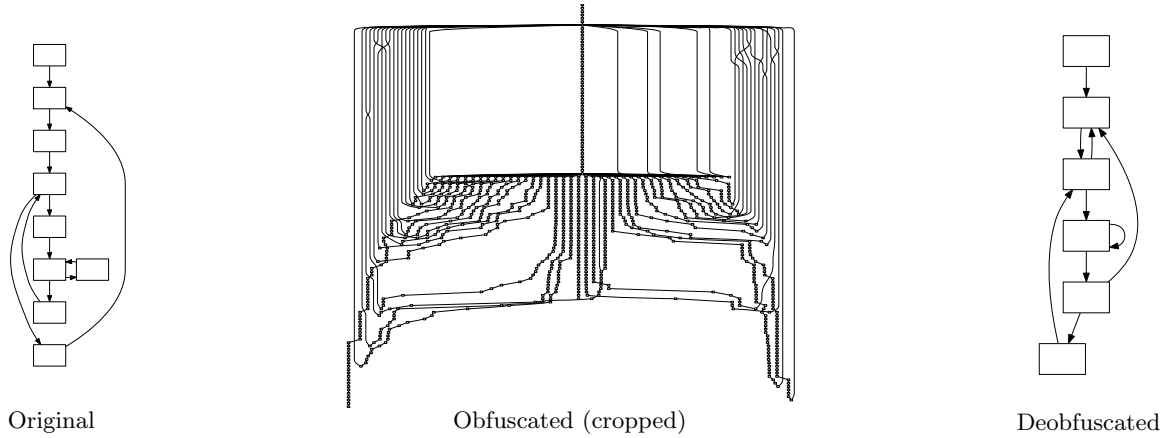
Original Obfuscated (cropped) Deobfuscated

(*a*) Netsky_ae1: Code Virtualizer



Original Obfuscated (cropped) Deobfuscated

(*b*) Hunatcha: ExeCryptor



Original Obfuscated (cropped) Deobfuscated

(*c*) Matrix multiply: Themida

Figure 4: Effects of obfuscation and deobfuscation on the control flow graphs of some malware samples

1. *Semantics-based approaches to identifying metamorphic malware* [1].

   Metamorphic code includes self-modifying semantics-preserving transformations to exploit code diversification. The impact of metamorphism is growing in security and code protection technologies, both for preventing malicious host attacks, e.g., in software diversification for IP and integrity protection, and in malicious software attacks, e.g., in metamorphic malware self-modifying their own code in order to foil detection systems based on signature matching. In this paper we consider the problem of automatically extracting metamorphic signatures from metamorphic code. We introduce a semantics for self-modifying code, later called phase semantics, and prove its correctness by showing that it is an abstract interpretation of the standard trace semantics. Phase semantics precisely models the metamorphic code behavior by providing a set of traces of programs which correspond to the possible evolutions of the metamorphic code during execution. We show that metamorphic signatures can be automatically extracted by abstract interpretation of the phase semantics. In particular, we introduce the notion of regular metamorphism, where the invariants of the phase semantics can be modeled as finite state automata representing the code structure of all possible metamorphic change of a metamorphic code, and we provide a static signature extraction algorithm for metamorphic code where metamorphic signatures are approximated in regular metamorphism.

References

[1] Mila Dalla Preda, Roberto Giacobazzi, and Saumya Debray. Unveiling metamorphism by abstract interpretation of code properties. *Theoretical Computer Science*, 577:74–97, April 2015.

[2] G. Lu and S. Debray. Automatic simplification of obfuscated JavaScript code: A semantics-based approach. In *Proc. Sixth IEEE International Conference on Software Security and Reliability (SERE 2012)*, pages 31–40, June 2012.

[3] Gen Lu. *Analysis of Evasion Techniques in Web-Based Malware*. PhD thesis, University of Arizona, January 2014.

[4] Gen Lu, Karan Chadha, and Saumya Debray. A simple client-side defense against environment-dependent malware. In *Proc. 8th International Conference on Malicious and Unwanted Software (The Americas)*, October 2013.

[5] Gen Lu and Saumya Debray. Weaknesses in defenses against web-borne malware (extended abstract). In *Proc. 10th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, July 2013.

[6] Jing Qiu, Babak Yadegari, Brian Johannesmeyer, Saumya Debray, and Xiaohong Su. Identifying and understanding self-checksumming defenses in software. In *Proc. Fifth ACM Conference on Data and Application Security and Privacy (CODASPY)*, March 2015.

[7] Babak Yadegari. *A Generic Approach to Deobfuscation*. PhD thesis, University of Arizona. In preparation.

[8] Babak Yadegari and Saumya Debray. Bit-level taint analysis. In *IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 2014.

[9] Babak Yadegari, Brian Johannesmeyer, Benjamin Whitely, and Saumya Debray. A generic approach to automatic deobfuscation of executable code. In *Proc. 36th IEEE Symposium on Security and Privacy*, May 2015.

1.

1. Report Type

Final Report

Primary Contact E-mail

Contact email if there is a problem with the report.

debray@cs.arizona.edu

Primary Contact Phone Number

Contact phone number if there is a problem with the report

5206214527

Organization / Institution name

University of Arizona

Grant/Contract Title

The full title of the funded effort.

Tools for Rapid Understanding of Malware Code

Grant/Contract Number

AFOSR assigned control number. It must begin with "FA9550" or "F49620" or "FA2386".

FA9550-11-1-0191

Principal Investigator Name

The full name of the principal investigator on the grant or contract.

Saumya Debray

Program Manager

The AFOSR Program Manager currently assigned to the award

Robert Herklotz

Reporting Period Start Date

07-15-2011

Reporting Period End Date

03-14-2015

Abstract

The project goals were to develop automated techniques and tools for analysis and reverse engineering of highly-obfuscated malware codes. The project made significant progress in this regard. Two very different kinds of malware were considered; because of the different nature of the malicious code in each case, fundamentally different techniques were employed. The two kinds of malware, and specific accomplishments for each, are listed below.

1. Web-delivered malware, which is typically in the form of (obfuscated) JavaScript programs. This kind of malware is currently among the commonest way for infections to occur. Specific accomphishments include: development of novel techniques for reverse engineering obfuscated JavaScript code; identification of weaknesses in existing techniques for detecting web-borne malware; and development of client-side defenses against trigger-based web-based malware.

2. Native executables, whcich refer to machine code programs that execute natively.

Regardless of whether an infection happened through the web via JavaScript code or not, in the end the malicious actions are typically carried out by native executables. Specific accomplishments include: development of improved techniques for information flow analysis in software; generic techniques for deobfuscation of executable code; and information flow based dynamic analysis techniques for identifying self-checksum-based anti-tamper defenses in software.

The project led to two PhD dissertations:

1. Gen Lu. Analysis of Evasion Techniques in Web-Based Malware. PhD Dissertation, The University of Arizona, Jan. 2014. This dissertation focused on analysis of web-based malware. The significant accomplishments of this work are listed above.

2. Babak Yadegari. A Generic Approach to Deobfuscation. PhD Dissertation, The University of Arizona, May 2016 (expected).
This dissertation focused on analysis of native malware. The significant accomplishments of this work are listed above.

Software and data samples resulting from the project are available to the research community at
http://www.cs.arizona.edu/projects/lynx/Samples/.

Distribution Statement

This is block 12 on the SF298 form.

Distribution A - Approved for Public Release

Explanation for Distribution Statement

If this is not approved for public release, please provide a short explanation. E.g., contains proprietary information.

SF298 Form

Please attach your SF298 form. A blank SF298 can be found here. Please do not password protect or secure the PDF The maximum file size for an SF298 is 50MB.

AFD-070820-035.pdf

Upload the Report Document. File must be a PDF. Please do not password protect or secure the PDF . The maximum file size for the Report Document is 50MB.

body.pdf

Upload a Report Document, if any. The maximum file size for the Report Document is 50MB.

Archival Publications (published) during reporting period:

[1] Mila Dalla Preda, Roberto Giacobazzi, and Saumya Debray. Unveiling metamorphism by abstract interpretation of code properties. Theoretical Computer Science, 577:74–97, April 2015.
[2] G. Lu and S. Debray. Automatic simplification of obfuscated JavaScript code: A semantics-based approach. In Proc. Sixth IEEE International Conference on Software Security and Reliability (SERE 2012), pages 31–40, June 2012.
[3] Gen Lu. Analysis of Evasion Techniques in Web-Based Malware. PhD thesis, University of Arizona, January 2014.
[4] Gen Lu, Karan Chadha, and Saumya Debray. A simple client-side defense against environment-dependent malware. In Proc. 8th International Conference on Malicious and Unwanted Software (The Americas), October 2013.
[5] Gen Lu and Saumya Debray. Weaknesses in defenses against web-borne malware (extended abstract). In Proc. 10th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA), July 2013.

[6] Jing Qiu, Babak Yadegari, Brian Johannesmeyer, Saumya Debray, and Xiaohong Su. Identifying and understanding self-checksumming defenses in software. In Proc. Fifth ACM Conference on Data and Application Security and Privacy (CODASPY), March 2015.
[7] Babak Yadegari. A Generic Approach to Deobfuscation. PhD thesis, University of Arizona. In preparation.
[8] Babak Yadegari and Saumya Debray. Bit-level taint analysis. In IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM), 2014.
[9] Babak Yadegari, Brian Johannesmeyer, Benjamin Whitely, and Saumya Debray. A generic approach to automatic deobfuscation of executable code. In Proc. 36th IEEE Symposium on Security and Privacy, May 2015.

Changes in research objectives (if any):

None.

Change in AFOSR Program Manager, if any:

Dr. Robert Herklotz, the Program Manager for this project, retired in 2014.

Extensions granted or milestones slipped, if any:

None.

AFOSR LRIR Number

LRIR Title

Reporting Period

Laboratory Task Manager

Program Officer

Research Objectives

Technical Summary

Funding Summary by Cost Category (by FY, $K)

	Starting FY	FY+1	FY+2
Salary			
Equipment/Facilities			
Supplies			
Total			

Report Document

Report Document - Text Analysis

Report Document - Text Analysis

Appendix Documents

2. Thank You

E-mail user

May 04, 2015 20:03:34 Success: Email Sent to: debray@cs.arizona.edu